(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) W rld Intellectual Property Organization
International Bureau

(43) International Publication Date
14 March 2002 (14.03.2002)

PCT

(10) International Publication Number
WO 02/21339 A2

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/US01/28180

(22) International Filing Date: 7 September 2001 (07.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/230,878	7 September 2000 (07.09.2000)	US
Not furnished	6 September 2001 (06.09.2001)	US
Not furnished	6 September 2001 (06.09.2001)	US
Not furnished	6 September 2001 (06.09.2001)	US

(71) Applicant: ORACLE CORPORATION [US/US]; 500 Oracle Parkway, Redwood Shores, CA 94065 (US).

(72) Inventors: KRISHNAPRASAD, Muralidhar; 1065 Foster City Boulevard, Unit D, Foster City, CA 94404 (US). KRISHNAMURTHY, Viswanathan; 4735 Touchstone Terrace, Fremont, CA 94555 (US). MURTHY, Ravi; 2493 Creekside Court, Hayward, CA 94542 (US). NIMANI, Visar; 409 Luss Lane, Redwood City, CA 94065 (US).

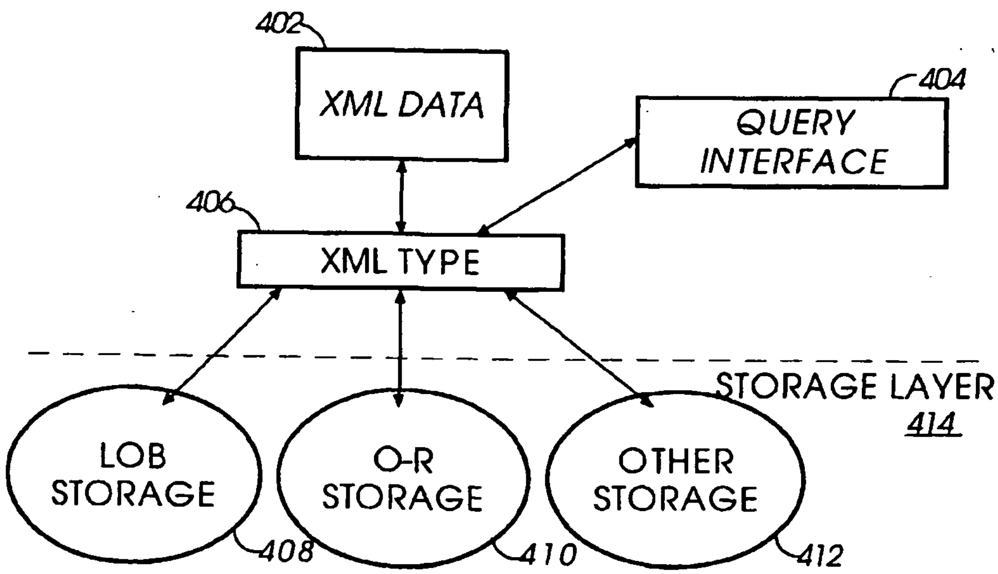
(74) Agents: TAN, Carina et al.; HICKMAN PALERMO TRUONG & BECKER, LLP, 1600 Willow Street, San Jose, CA 95125 (US).

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR XML DATA STORAGE, QUERY REWRITES, VISUALIZATION, MAPPING AND REFERENCES



(57) Abstract: Techniques are provided for one or more portions of the relational database to visualized as an XML document. A standard Uniform Resource Locator (URL) mechanism is provided to access data stored in the relational database by defining the URL as an XPath expression over the visualized XML document. Techniques are provided for mapping XML data and metadata from data in relational databases, for allowing the user to use a database query to retrieve data from a relational database in the form of XML documents, and for XML data storage and query rewrites in relational databases.

WO 02/21339 A2

IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR XML DATA STORAGE, QUERY REWRITES, VISUALIZATION, MAPPING AND REFERENCING

RELATED APPLICATIONS; PRIORITY CLAIM

This application claims priority from U.S. Provisional Patent Application Serial Number 60/230,878 filed on September 7, 2000 entitled "XML DATA STORAGE, QUERY REWRITES, VISUALIZATION, MAPPING AND REFERENCING", by inventors Muralidhar Krishnaprasad, Viswanathan Krishnamurthy, and Ravi Murthy.

This application claims priority from U.S. patent application serial number __________, Attorney Docket No. 50277-1564, entitled "METHOD AND APPARATUS FOR XML VISUALIZATION OF A RELATIONAL DATABASE AND UNIVERSAL RESOURCE IDENTIFIERS TO DATABASE DATA AND METADATA," naming as inventors Muralidhar Krishnaprasad, Viswanathan Krishnamurthy, Ravi Murthy.

This application claims priority from U.S. patent application serial number __________, Attorney Docket No. 50277-1565, entitled "APPARATUS AND METHOD FOR MAPPING RELATIONAL DATA AND METADATA TO XML," naming as inventors Muralidhar Krishnaprasad, Viswanathan Krishnamurthy, Ravi Murthy and Visar Nimani.

This application claims priority from U.S. patent application serial number __________, Attorney Docket No. 50277-1566, entitled "METHOD AND APPARATUS FOR FLEXIBLE STORAGE AND UNIFORM MANIPULATION OF XML DATA IN A RELATIONAL DATABASE SYSTEM", naming as inventors Muralidhar Krishnaprasad, Viswanathan Krishnamurthy, and Ravi Murthy.

FIELD OF THE INVENTION

The present invention relates generally to relational databases and, more specifically, to XML visualization of a database, DBURI references to database objects, mapping relational database data and metadata to XML data, providing XML data in response to XML queries, and XML data storage, manipulation and queriability.

BACKGROUND OF THE INVENTION

On the World Wide Web, there is a need to reference data from different sources inside documents. The standard way of referencing such data is through the use of URIs – or universal resource identifiers. Since a vast majority of the data lies inside relational

databases, it is necessary to support a standard URI based access methods to such data. Typically, such applications are written using standard mechanisms like Servlets, which in-turn may execute SQL statements to retrieve and format the database data. Significant processing is often needed to convert the results of the SQL data into a standard format required by the user, such as extensible Markup Language (XML). XML is a World Wide Web Consortium (W3C) standard for representing data.

Data within relational databases are typically accessed by sending commands to database servers that manage the databases. Such commands must conform to a database language supported by the database server.

Many applications are now being designed to expect input data in the form of XML documents. When the data provided to the applications comes from a relational database, the data typically has to be reformatted into an XML document.

When data is presented as an XML document, the recipient of the document must understand the structure of the XML document. When the XML document is generated from the result of a relational database query, the structure of resulting XML document will typically vary based on the nature of the query. Consequently, the process of transforming the relational data to an XML document, and generating data that indicates the structure of the XML document thus produced, can be cumbersome and inflexible.

Various techniques may be used to store data from such XML documents into a relational database. According to one technique, each XML document is treated as a single data item, and stored as such in a single column of a relational table. This technique is convenient in that the XML does not have to be processed before it is submitted to the database server. However, because the database server considers the XML document a single data item, the database server is unable to take advantage of the fact that XML documents are structured, where a single XML document may include numerous attributes and elements with specific values.

According to an alternative technique, an XML document may be split up into its constituent attributes and element data before the XML document is stored in a database. The values for each attribute and element are submitted to the database for insertion in corresponding columns of a table. When this technique is used, the database server may be used to select data based on individual attribute values. However, when the data is retrieved from the database, the attribute values are provided as distinct data items, not as part of a single XML document. To recover the XML document, the data received from the database server must be reformatted and structured to reconstruct the XML document.

Based on the foregoing, it is clearly desirable to provide a less cumbersome mechanism and technique for allowing clients, such as browsers, that support accessing resources using URLs, to access relational data.

Further, it is desirable to provide techniques for transforming relational data into XML in an intelligent, flexible and efficient manner.

It is desirable to provide techniques for handling XML documents that do not depend on the specific form in which they are stored. In other words, it is desirable for applications to independently decide on the storage representation of their XML data, and that does not have any impact on the functionality. However, the choice of storage could potentially affect performance of the applications. Further it is also desirable for the database server to implement techniques that exploit the chosen storage representation for optimal processing of user operations.

SUMMARY OF THE INVENTION

Techniques are provided for allowing a user to view and retrieve data from a relational database in XML format. Techniques are provided for (1) the user to access this data through the World Wide Web by providing URI references to such data, and (2) to store and perform operations on these URI references inside the database.

Techniques are also provided for using XML syntax in exchanging data with relational databases. Users may navigate through those "visualized" portions of the database using XPath expressions.

Techniques are provided for mapping metadata and data in relational databases to XML data. According to certain embodiments of the invention, a mechanism is provided to allow the user to use a database query to retrieve data from a relational database in the form of XML documents by canonically mapping object relational data to XML data and canonically mapping object relational schemas to XML-Schemas. XML Namespaces are used to augment the schema information, by mapping database metadata objects in different database schemas to different XML namespaces.

Techniques are provided for modeling XML data using an abstract data type in a relational database system.

A mechanism is provided to generate a database query based on an XML query and the mapping information when a user submits the XML query to access the data in the XML document that is stored in the relational database. This process involves

rewriting the user queries (and other data manipulation operations) into other queries that better exploit the underlying storage representation of the XML data.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. 1A is a block diagram that illustrates schemas and schema objects in a relational database;

FIG. 1B is a block diagram that illustrates tables stored in a relational database;

FIG. 2A is a block diagram that illustrates a hierarchy of Uritypes;

FIG. 2B is a block diagram that illustrates a relational database table that stores Uritype data;

FIG. 3A is a block diagram that illustrates a table;

FIG. 3B is a block diagram that illustrates a view that contains a SQL result set;

FIG. 4 is a block diagram that illustrates an XMLType storage architecture;

FIG. 5 is a block diagram that illustrates the structure of an XMLType column; and

FIG. 6 depicts a computer upon which embodiments of the invention may be implemented.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Techniques are provided for using XML syntax for accessing data in relational databases, for mapping data and metadata from relational databases to XML, for modeling XML data using an abstract data type in a relational database system, for multiple storage representations, for uniform query interface, and for optimal processing using query rewrites. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

FUNCTIONAL OVERVIEW

Using the techniques described herein, any particular user of the relational database, herein referred to as the "current user", can visualize as an XML tree all the tables and views and the associated schema, in the relational database, to which the current user has been granted access privileges. In other words, the user, instead of seeing the database data in the form of tables and views, the data is presented to the user in the form of an XML document, and the typical structure of an XML document is a tree.

There may be several concurrent current users of the database. However, for the purpose of simplifying the description, the techniques described herein refer to a single current user. The XML tree is hereafter referred to as a "visualized XML document". The visualized XML document comprises XML representations of the tables and views and the associated schema. Because the XML document is based on the access rights of a user, the XML document will vary from user to user based on each user's access rights. Thus, the visualized XML document, as described herein, is associated with the current user.

The data items that are identified by a URL or URI, and that are to be accessed in a relational database, are referred to herein as "target data". Target data may vary from implementation to implementation. Target data may be any one of numerous types of data, such as relational database schema objects, relational data, control files, etc. The present invention is not limited to target data of any particular type.

In order for the current user to access and manipulate target data in the relational databases as if the target data are XML data, a mechanism is provided for: 1) defining a default virtual visualization of any relational database for the current user, including all of the data for which the current user has access privileges in the relational database, wherein the default virtual visualization is defined as a canonical XML document, 2) providing a standard Uniform Resource Indicator (URI) that is locally defined within the database and by which one or more fragments of the visualized XML document can be accessed by defining the URI as an XPath expression over the visualized XML document, 3) providing a standard Uniform Resource Locator (URL) as a mechanism that can be used in conjunction with a web browser that is external to the relational database for accessing data stored in the relational database by defining the URL as an Xpath expression over the visualized XML document, 4) providing, in the relational database, new data types and new object types that can be used for storing the URIs and URLs, and

5) providing a mechanism to modify, create, add or delete data in the relational database through the visualized document, using the standard URIs and URLs as described herein.

XPath expressions are W3c standard ways of navigating XML documents. XPath expressions allow for traversals and insert/deletes/updates on the visualized XML document (the XML visualization of the relational database). The XPath expressions can be converted into SQL Data Definition Language (DDL) and Data Manipulation Language (DML) commands in the relational database.

The combination of (1) XML visualization of the relational database, and (2) the mechanism for navigating through the visualized XML document using XPath expressions, allows the user to "point to" any data in the relational database. For example, an XPath expression such as, /SCOTT/EMP/ROW[EMPNO=2100] points to the data value in the row identified by EMPNO =21. The row in question is in a relational database table called EMP, which is a schema object of the relational database schema called Scott. The XML visualization of the relational database is described in greater detail herein.

The results of a database query is herein referred to as a "SQL result set". When the SQL result set is converted into one or more XML documents, the converted result set is herein referred to as an "XML result set". The data in the relational database is herein referred to as "object relational data" and metadata that is associated with the object relational data is herein referred to as relational database schemas. A relational database schema is a collection of objects, herein referred to as "schema objects". Schema objects are logical structures that directly refer to the data in the relational database. Thus, schema objects include structures such as tables, views, clusters, indexes, etc.

In order that users may submit a query using relational database query language and receive a result set in the form of XML documents, a mechanism is provided for: 1) Mapping of object relational data from the relational database to XML form; 2) Mapping of the relational database schemas to XML form; 3) Generating XML documents from the object relational data and the relational database schemas.

The mechanism for generating XML documents as the result set from object relational data and relational database schemas is based on a set of rules stored in the relational database. Object relational data, either pre-defined or dynamically defined during the processing of the database query, are mapped to corresponding XML data based on the set of rules. Such a mapping is herein referred to as "canonical mapping of object relational data to XML form".

Similarly, relational database schemas are mapped to XML form by mapping the relational database schemas to their corresponding XML-Schemas. Such a mapping is based on a set of rules that are herein referred to as "canonical mapping of relational database schemas to XML-schemas." The canonical mapping of relational database schemas to XML-schemas is described in greater detail herein.

Further, generating an XML result set, i.e., generating XML documents from the SQL result set is based on a set of rules, as described in greater detail herein.

In order to integrate the handling of XML data and SQL data in a relational database, a mechanism is provided to support an XML type datatype in the relational database for storing XML documents within columns and rows of tables. The storage representation may vary from implementation to implementation. The present invention is not limited to any particular storage representation. In certain embodiments of the invention, the user may submit an XML document for storage in the relational database. The data from each field of the XML document is automatically stored in the relational database in a manner that leverages various existing indexing mechanisms associated with storage. For example, when an XML document is submitted for storage in the relational database, a mechanism is provided to determine the corresponding storage column in the relational database for storing each field of the XML document. Thus, each field of the XML document is mapped to some column in the relational database and this mapping information is stored in the relational database. Depending on the type of data, some of the fields of data may be lumped together for storage in a single object relational column while other fields may be stored as separate object relational columns. The data from each field of XML document can be indexed using an appropriate indexing scheme. For example, a B-tree index may be used for columns that contain relational type data and a text index, such as interMedia text, may be used for columns that contain large text data. In certain embodiments, the user may specify the mapping information. By specifying the mapping information, the user may control the granularity of the mapping.

Thus, techniques are provided for 1) the uniform handling of XML data and SQL data, 2) a uniform query interface for a well-defined set of XML operations, where the set of operations are decoupled from the underlying storage mechanism for the XML data in the relational database, 3) query rewrites into a form that leverages the data access and data manipulation capabilities of the underlying storage mechanism in the relational database.

VIRTUAL XML VISUALIZATION OF A RELATIONAL DATABASE AND XPATH

According to certain embodiments, the current user can see all the data in the relational database, to which the user has been granted access privileges, as a visualized XML document. The visualized XML document includes a "database tag" and a set of schemas with tables and views. For example, if the database is called "oradb" then the XML document starts with the database tag "<oradb>" and ends with the databse tag "</oradb>".

The current user is permitted to read, insert, update and delete elements from the visualized XML document. The current user is thus unaware of the actual nature of the storage or access of the data in the relational database. The current user simply uses XPath expressions for navigation through the visualized XML document.

For example, assume that the current user of the relational database is Scott. Associated with each current user of the relational database is a schema by the same name. A schema is a logical collection of relational database objects such as tables, views clusters, functions, etc.

FIG. 1A is a block diagram that illustrates schemas and schema objects in a relational database. Column 102 of FIG. 1A contains a list of schema objects. For the purpose of explanation, only two schemas, SCOTT and JONES, are shown. Column 104 contains the relational database objects in each schema. . For the purpose of explanation, only two tables, EMP and DEPT, are shown. Schema SCOTT contains table EMP and schema JONES contains table DEPT.

FIG. 1B is a block diagram that illustrates tables stored in a relational database. Relational database 110 contains tables, EMP and DEPT. Table EMP has 3 columns, EMPNO, ENAME and SALARY. Each column of EMP contains rows of values. Table DEPT has 2 columns, DEPTNO and DNAME. Each column of DEPT contains rows of values.

Assume that the current user, Scott, has privileges to access schemas SCOTT and JONES in the relational database and has privileges to access the data associated with SCOTT and JONES. According to certain embodiments of the techniques, the current user, Scott, is able to see a default virtual visualization of the relational database as follows (not the full visualization):

```
<oradb>
  <SCOTT>
    <EMP>
      <ROW>
        <EMPNO>21</EMPNO>
        <ENAME>John</ENAME>
        <SALARY>100,000</SALARY>
      </ROW>
       <ROW>
         .... <!-- other emp values -->
       </ROW>
       ...
     </EMP>
      .... <!-- other tables.. -->
  </SCOTT>
  <JONES>
     <DEPT>
      <ROW>
        <DEPTNO>1</DEPTNO>
        <DNAME>Marketing</DNAME>
      </ROW>
       <ROW>
         .... <!-- other dept values -->
       </ROW>
     </DEPT>
 </JONES>
</oradb>
```

The above default virtual visualization is merely an example of one implementation of a default virtual visualization. The default virtual visualization may vary from implementation to implementation. The invention is not restricted to a particular visualization model.

According to certain embodiments of the techniques, a standard URL and URI mechanism is also provided to access the data stored in any database, by defining the URL and URI as an XPath expression over the visualized XML document.

According to one embodiment of the invention, the URL may be processed by using a servlet, which uses the native URI processing mechanisms to access the data pointed by the URL.

According to certain embodiments of the invention, the database tag (oradb) may be implicitly bound in the processing context and need not be explicitly specified in the URL.

A current user who does not have local access to the relational database may use a browser to access data in the relational database over the Internet by using a URL. For

example, assume that the current user is Scott and that Scott would like to use a browser to access the employee-name column of EMP table at the row for which the employee number is 2100, and where the EMP table is in schema SCOTT. The URL that Scott would use may look like the following:

http://machine-name/servlet-name/SCOTT/EMP/ROW[EMPNO=2100]/ENAME

In the above URL, the database tag, "oradb", is implicitly bound, and thus the user need not specify the database tag in the URL.

The result of accessing the URL or the URI would be a fragment of the visualized XML document containing the ename argument as shown below:

```
<? xml version="1.0"?>
<ENAME> John</ENAME>
```

The current user can augment the URL or URI with content types to specify the Multipurpose Internet Mail Extensions (MIME) type of the output. For example, if the URL points to a BLOB (Binary Large OBject) column that is storing an image, wherein the image is the "target data", then the content type can be set to gif. Thus, in response to using the URL, the current user obtains the image rather than, for example, a large hexadecimal file.

As another example, the current user may augment the URL to request, as the target data, the text value of the column to which the URL points. For example, assume that the current user, Scott, uses the following URL to access the employee-name column of EMP table at the row for which the employee number is 2100:

http://machine-name/<servlet-name>/
/SCOTT/EMP/ROW[EMPNO=2100]/ENAME/text()

"text()" is an XPath standard to identify the text node. The use of text() in the above URL would produce a result that contains only the text value in the employee-name column of EMP table at the row for which the employee number is 2100. The text value in the employee-name column of EMP table at the row for which the employee number is 2100 is "John". Thus, the result of accessing the above URL using text() is "John". In contrast, when text() is not used in the URL to access the employee-name column in the example, "John" is inlined inside a fragment of the visualized XML document as follows:

<ENAME> John</ENAME>

In another embodiment of the invention, the mime information may be derived automatically by the database, based on other auxiliary information that may be stored with the URL or through user written functions.

MAPPING RULES FOR DEFINING VIRTUAL XML VISUALIZATION OF A RELATIONAL DATABASE

Techniques are provided for defining the default virtual visualization of the database as a canonical XML document. According to one embodiment, the rules for defining the default virtual visualization are as follows:

1) There is a pseudo top-level enclosing tag that identifies the relational database that contains the target data. An example of a pair of enclosing tags that identify the relational database that contains the target data is <oradb> ...</oradb>, where "oradb" is the name of the relational database that contains the target data, as shown below (not all the elements in the visualization are shown):

```
<oradb>
  <SCOTT>
     <EMP>
       <ROW>
         <EMPNO>21</EMPNO>
         <ENAME>John</ENAME>
         <SALARY>100,000</SALARY>
       </ROW>
       ...
     </EMP>
     .... <!-- other tables.. -->
  </SCOTT>
  <JONES>
     <DEPT>
       <ROW>
         .... <!-- other dept values -->
       </ROW>
     </DEPT>
  </JONES>
</oradb>
```

2) Each schema in the relational database for which the current user has been granted access privileges corresponds to one element in the visualized XML document. The name of the element is the same as the name of the schema to which the element corresponds. In the example illustrated herein, the schema SCOTT in the relational database "oradb" is represented by the element with the same name in the visualized XML document. Similarly, the schema JONES is represented by the element with the same name in the visualized XML document. The following visualized XML document is a visualization of the relational database down to the schema element level. For the purpose of illustration, only the elements corresponding to schemas SCOTT and JONES are shown.

```
<oradb>
  <SCOTT>
   </SCOTT>
   <JONES>
  </JONES>
 </oradb>
```

3) Each table or view in the relational database for which the current user has been granted access privileges corresponds to one element in the visualized XML document. The name of the element is the same as the name of the table or view to which the element corresponds. In the example illustrated herein, the table EMP in the relational database "oradb" is represented by the element with the same name in the visualized XML document. Similarly, the table DEPT is represented by the element with the same name in the visualized XML document. The following visualized XML document is a visualization of the relational database down to the table element level. For the purpose of illustration, only the elements corresponding to tables EMP and DEPT are shown.

```
<oradb>
  <SCOTT>
      <EMP>
      </EMP>
   </SCOTT>
   <JONES>
      <DEPT>
      </DEPT>
  </JONES>
 </oradb>
```

4) Each row of each table or view in the relational database for which the current user has been granted access privileges corresponds to one element in the visualized XML document. The following visualized XML document is a visualization of the relational database down to the row element level. For the purpose of illustration, only the elements corresponding to rows of tables EMP and DEPT are shown.

```
<oradb>
  <SCOTT>
    <EMP>
      <ROW>
      </ROW>
       ... <!—multiple ROW tag pairs -  each pair corresponding to a single row>
    </EMP>
  </SCOTT>
  <JONES>
    <DEPT>
      <ROW>
      </ROW>
       ... <!—multiple ROW tag pairs -  each pair corresponding to a single row>
    </DEPT>
  </JONES>
</oradb>
```

5) Each column of each table or view in the relational database for which the current user has been granted access privileges corresponds to one element in the visualized XML document. The name of the element is the same as the name of the column to which the element corresponds. In the example illustrated herein, the column EMPNO in table EMP is represented by the element with the same name in the visualized XML document. Similarly, the column ENAME is represented by the element with the same name in the visualized XML document. The following visualized XML document is a visualization of the relational database down to the column element level. For the purpose of illustration, only the elements corresponding to columns EMPNO and ENAME in table EMP are shown.

```
<oradb>
  <SCOTT>
    <EMP>
      <ROW>
        <EMPNO>2100</EMPNO>
        <ENAME>JOHN</ENAME>
        <SALARY>100000</SALARY>
      </ROW>
      <ROW>
        <EMPNO>2200</EMPNO>
        <ENAME>TOM</ENAME>
        <SALARY>35000</SALARY>
      </ROW>
        .... <!—other rows..>
    </EMP>
  </SCOTT>
  <JONES>
    <DEPT>
      <ROW>
        <DEPTNO>1</DEPTNO>
        <DNAME>MARKETING</DNAME>
      </ROW>
        ... <!—other rows..>
    </DEPT>
  </JONES>
</oradb>
```

RULES FOR CONVERTING XPATH EXPRESSIONS INTO RELATIONAL DATABASE QUERIES

According to one embodiment of the invention, the Xpath query over the XML visualization may be translated into relational database queries and the result formatted in XML. Techniques are provided for converting XPath expressions into relational database queries. For the purpose of explanation, assume that the XPath expression that is to be converted into a query is in the context of the relational database "oradb". Thus, the format of a typical XPath expression is the context of "oradb" can be generalized as follows:

/Schema/Table/Row/Column/Attribute ...(! further attributes)

Each element of the above XPath expression may optionally have a predicate. Predicates take the form,

[(element) (operator) (value)]

For example, the element, Row, may have a predicate as follows,

/Schema/Table/Row [Column1 = 2100]/Column2

The rules for converting XPath expressions into relational database queries will refer to the above general format for an XPath expression and are as follows:

1) An XPath expression of the form /Schema/Table is converted to a corresponding relational database query such as the following SQL statement,

```
Select *
From Schema.Table
```

The syntax used in the preceding statement is merely illustrative. The actual syntax of SQL statements may vary from implementation to implementation. The present invention is not limited to any particular syntax.

The results of the SQL statement can then be converted into a corresponding fragment of the visualized XML document.

2) An XPath expression of the form /Schema/Table/Row/Column is converted to a corresponding relational database query such as the following SQL statement,

```
Select Column
From Schema.Table
```

The syntax used in the preceding statement is merely illustrative. The actual syntax of SQL statements may vary from implementation to implementation. The present invention is not limited to any particular syntax.

The results of the SQL statement can then be converted into a corresponding fragment of the visualized XML document.

3) An XPath expression of the form /Schema/Table/Row [Empno = 2100]/Column is converted to a corresponding relational database query such as the following SQL statement,

```
Select Column2
From Schema.Table
Where Column1 = 2100
```

The syntax used in the preceding statements is merely illustrative. The actual syntax of the query language statements may vary from implementation to implementation. The present invention is not limited to any particular syntax.

The results of the query language statement can then be converted into a corresponding fragment of the visualized XML document.

URI TYPES

According to certain embodiments, a special data type is provided in the relational database for storing URIs and URLs in the relational database. Such a data type is herein referred to as an "Uritype". URIs and URLs can be stored in columns in relational database tables by defining the URIs and URLs as Uritype data.

FIG. 2A is a block diagram that illustrates a hierarchy of Uritypes. FIG. 2A shows a general Uritype 210 that comprises subtypes. According to certain embodiments, the subtypes are DB-Uritype 212, HTTP-Uritype 214, and FTP-Uritype 216.

The HTTP-Uritype stores HTTP (HyperText Transfer Protocol) URLs, and fetches the data pointed to by the URL using the HTTP protocol. The FTP-Uritype stores FTP (File Transfer Protocols) URLs and fetches the data, using FTP. The DB-Uritype stores intra-database references using the Xpath mechanism described herein.

The DB-Uritype can fetch the data associated with the URL using the Xpath translation mechanism defined earlier or through other mechanisms.

The user may define subtypes of the Uritype or any of the other subtypes, and provide the implementation for getting the data pointed to by that URL.

Apart from being able to store URIs and URLs, the general functions associated with the Uritype data type include retrieving URIs and URLs as well as retrieving XML documents that are stored as LOBs, for example, CLOBs and BLOBs, in the relational database.

When the current user wishes to retrieve target data, pointed to by the URL, from the relational database, the current user's XPath expressions are automatically converted into appropriate query language statements. The actual syntax of such statements depend on the query language used in the relational database and may vary from implementation to implementation. The present invention is not limited to any particular syntax. The relational database functions of the Uritype data type can be abstracted by the following statements:

```
getURL( );
getBLOB( );
getCLOB( ).
getXML();
```

The preceding statements merely illustrate function. Such statements are not necessarily query language statements. The invention is not restricted to a particular set of query language statements.

FIG. 2B is a block diagram that illustrates a relational database table that stores Uritype data. Relational database table 200 is a Purchase Order table. Table 200 has 2 columns: Purchase Order Number column 250 and Purchase Order Link column 260. Both columns 250 and 260 contain 3 rows of data, namely, row 271, row 272 and row 273. The Purchase Order Link column can store data of type UriType.

Column 260 at row 271 stores data of type HTTP-Uritype. Column 260 at row 272 stores data of type FTP-Uritype. Finally, column 260 at row 273 stores data of type DB-Uritype. Note that since the DB-Uritype, HTTP-Uritype etc., have been defined as subtypes of the UriType type, we can store instances of these types in the Purchase Order Link column.

The current user can retrieve any of the data pointed by the Uritype data stored in table 200 The database will automatically fetch the data using the appropriate mechanisms.

For example, the query shown below, retrieves the Purchase Order data pointed to by the Purchase Order Link column:

```
Select p.PurchaseOrderLink.getXML()
From PurchaseOrderTable p;
```

The database will fetch the purchase order through HTTP for the first row, use FTP for the second row and use the DB-Uritype processing mechanism for the last row.

The syntax used in the preceding statements is merely illustrative. The actual syntax may vary from implementation to implementation. The present invention is not limited to any particular syntax. The conversion into the appropriate query is transparent to the current user. Thus, it is not necessary for the current user to be aware of the type of the target data.

MODIFYING RELATIONAL DATA USING URITYPE FUNCTIONS

According to certain embodiments, a mechanism is provided to modify, add or delete XML data that is stored in the relational database using the standard URIs and URLs as described herein.

For example, assume that the current user, Scott, is able to see a default virtual visualization of the relational database as follows (not the full visualization):

```
<oradb>
  <SCOTT>
    <EMP>
      <ROW>
        <EMPNO>21</EMPNO>
        <ENAME>John</ENAME>
        <SALARY>100,000</SALARY>
      </ROW>
       <ROW>
         .... <!-- other emp values -->
       </ROW>
       ...
     </EMP>
      .... <!-- other tables.. -->
   </SCOTT>
   <JONES>
     <DEPT>
      <ROW>
        <DEPTNO>1</DEPTNO>
        <DNAME>Marketing</DNAME>
      </ROW>
       <ROW>
         .... <!-- other dept values -->
       </ROW>
     </DEPT>
 </JONES>
</oradb>
```

Further assume that the current user, Scott, would like to update data at the employee-name column of EMP table at the row for which the employee number is 2100. The update comprises changing the name "John" to "Mary".

According to certain embodiments, if the current user, Scott, has direct access to the relational database, then Scott can perform the following: 1) selects the update operation for updating XML data and 2) uses the following XPath expressions:

```
/SCOTT/EMP/ROW[EMPNO=2100]/ENAME
<ENAME>Mary</ENAME>
```

The XPath expression /SCOTT/EMP/ROW[EMPNO=2100]/ENAME indicates the row and column of the target data that is to be updated.

The XPath expression <ENAME>Mary</ENAME> indicates the new value of the target data to be updated.

The above XPath expressions are converted into query language statements such as:

```
UPDATE "SCOTT"."EMP"
SET "ENAME" = 'Mary'
Where "EMPNO" = 2100;
```

If the current user, Scott, is using a web browser to access target data in the relational database, then according to certain embodiments, a general purpose servlet may be provided to allow the current user to modify, add or delete XML data that is stored in the relational database using the standard URIs and URLs. Using the above example of updating the name "John" to "Mary", a general purpose servlet is provided that allows Scott to perform the following: 1) select the update operation for updating XML data and 2) post to the servlet the "update" information in the form of the following XPath expressions:

```
/SCOTT/EMP/ROW[EMPNO=2100]/ENAME
<ENAME>Mary</ENAME>
```

According to certain other embodiments, a special servlet may be provided for each database operation. In other words, there may be an "Insert-servlet" for the INSERT operation, a "Delete-servlet" for the DELETE operation, and an "Update-servlet" for the UPDATE operation.

Using the above example of updating the name "John" to "Mary", an "Insert_servlet" is provided that allows Scott to perform the following: 1) select the update operation for updating XML data and 2) post to the servlet the "update" information in the form of the following XPath expressions:

```
http://machine-name/<update-
servlet>/SCOTT/EMP/ROW[EMPNO=2100]/ENAME
<ENAME>Mary</ENAME>
```

The same mechanism is used to modify, add or delete metadata. For example, if the current user, Scott would like to delete the schema SCOTT, then Scott can perform

the following: 1) selects the delete operation for deleting XML data in the relational database and 2) uses the following XPath expression that indicates which level in the visualized XML document to delete:

http://machine-name/<delete-servlet>/ SCOTT
XML RESULT SET FROM SQL RESULT SET

FIG. 3A is a block diagram that illustrates a table that is stored in a relational database. Table EMP has 3 columns, EMPNO, ENAME and SALARY. Each column of EMP contains a value for each of a plurality of rows of table EMP. However, for simplicity, FIG. 3A shows only 2 rows, namely, row 302 and row 304. In column EMPNO, row 302 contains the value "2100" and row 304 contains the value "2200". Similarly, in column ENAME, row 302 contains the value "JOHN" and row 304 contains the value "MARY". In column SALARY, row 302 contains the value "55K" and row 304 contains the value "65K".

As an example, assume that a user submits a relational database query to retrieve values from two of the columns, namely, EMPNO and ENAME from table EMP. An example of the relational database query is the following SQL statement,

SELECT empno, ename FROM emp

The syntax used in the preceding statement is merely illustrative. The actual syntax of SQL statements may vary from implementation to implementation. The present invention is not limited to any particular syntax.

FIG. 3B is a block diagram that illustrates a view that contains the SQL result set of the SQL statement, SELECT empno, ename FROM emp. The SQL result set comprises only two of columns of table EMP. Thus, FIG. 3B shows only columns EMPNO and ENAME. Even though the SQL result set comprises all the rows of value in columns EMPNO and ENAME, for simplicity, only rows 310 and 312 (which respectively contain values from rows 302 and 304) are shown in FIG. 3B.

The SQL result set of FIG. 3B can be converted into a corresponding XML result set based on the following rules, according to certain embodiments of the invention:

1) The XML result set is in the form of an XML document that begins with a ROWSET tag. The ROWSET tag indicates that the XML document comprises a set of row elements.

2) Each row of the SQL result set is converted to a corresponding ROW element in the XML document where the ROW element is indicated by a pair of ROW tags.

3) Each column within a given row of the SQL result set is converted to a corresponding COLUMN element that is embedded in the encompassing ROW element that in turn corresponds to the given row of the SQL result set. The name of the COLUMN tag is the name of the corresponding column in the SQL result set.

The ROWSET, and ROW tags may have varying monikers in various implementations of the invention. The present invention is not limited to any particular moniker for such tags. As for COLUMN tags, according to one embodiment, an aliasing mechanism is provided for changing names of the COLUMN tags.

To illustrate, the XML result set of the above example may appear like the following:

```
<ROWSET>
   <ROW>
      <EMPNO>21OO</EMPNO>
      <ENAME>John</ENAME>
   </ROW>
      <EMPNO>22OO</EMPNO>
      <ENAME>Mary</ENAME>
   <ROW>
        .... <!-- other column elements -->
   </ROW>
    ... <!-- other row elements -->
</ROWSET>
```

MAPPING OF OBJECT RELATIONAL DATA

An example of the canonical mapping of object relational data to XML form is as follows:

a) The mapping of object relational columns to XML elements:

For example, referring to FIG. 3A, the XML result set of a SQL query to select EMPNO and ENAME at the single row 302 would appear like the following:

```
<ROWSET>
   <ROW>
      <EMPNO>21OO</EMPNO>
      <ENAME>John</ENAME>
   </ROW>
</ROWSET>
```

The object relational columns EMPNO and ENAME are mapped to corresponding COLUMN elements in the XML result set. EMPNO and ENAME are embedded in the encompassing ROW element.

b) The mapping of object relational object types to XML elements with nested sub-elements containing the attributes of the object type:

For the purpose of explanation, assume that a user-defined type called "Address_t" is previously created in the relational database. "Address_t" is a complex object type that comprises 3 scalar attributes, namely CITY, STATE and ZIP. Assume that there is a table in the relational database called Employee_table. Employee_table has 2 columns ENAME and ADDRESS. ENAME is of type "string" and ADDRESS is the user-defined type "Address_t". Further assume that columns ENAME and ADDRESS each contain only a single row of value, namely, employee name "John" and John's address, respectively.

A SQL query such as "SELECT * FROM Employee_table" can produce a SQL result set that maps to the following XML result set, according to certain embodiments of the invention:

```
<ROWSET>
    <ROW>
      <ENAME>John</ENAME>
       <ADDRESS>
          <CITY>Redwood City</CITY>
          <STATE>California</STATE>
          <ZIP>94065</ZIP>
       </ADDRESS>
    </ROW>
</ROWSET>
```

The object relational column ADDRESS maps to an element with the same tag name in the XML result set. The element ADDRESS, which is of type "address_t", has attributes CITY, STATE and ZIP that are mapped to sub-elements of the element ADDRESS. Each sub-element has the name of the attribute as its corresponding tag name in the XML result set.

c) The mapping of object relational collection types to XML lists:

For the purpose of explanation, assume that a user-defined type called "Lineitems_t" is previously created in the relational database. "Lineitems_t" is a collection object type that comprises a plurality of collection items. Each collection item

has an ID attribute that defines the positional value of the item within the collection. For example, assume that there is a table in the relational database called PurchaseOrder_table. PurchaseOrder_table has 2 columns PONO for purchase order number and LINEITEMS for a list of line-items. PONO is of type "number" and LINEITEMS is of type "Lineitems_t". Further assume that columns PONO and LINEITEMS each contain a single row of value, namely, purchase order number "101" and the collection of lineitems that is associated with purchase order number "101". Assume that there is 1 item, namely, "books", in the collection of lineitems.

A SQL query such as "SELECT * FROM PurchaseOrder_table" produces a SQL result set that maps to the following XML result set:

```
<ROWSET>
  <ROW>
    <PONO>101</PONO>
    <LINEITEMS>
      <LINEITEMS_T id = "1">
        <LINEITEMNAME>BOOKS</ LINEITEMNAME>
        <COST>$150</COST>
      </LINEITEMS_T>
    </LINEITEMS>
  </ROW>
</ROWSET>
```

The object relational column LINEITEMS maps to an element with the same tag name in the XML result set. Each collection item in LINEITEMS is mapped to a sub-element that is embedded within the LINEITEMS element. Each collection item has the collection type name as the tag name, namely, "LINEITEM_T". Each collection item has attributes LINEITEMNAME, and COST that are mapped to sub-elements that are embedded within the element LINEITEM_T. Each sub-element has the name of the attribute as its corresponding tag name, e.g., <LINEITEMNAME>, and <COST>.

d) The mapping of object relational REF types to URI references:

Object relational REF columns are columns for storing object references to row objects contained in any object relational table. For the purpose of explanation, assume that the relational database has a table called Employee_table. Employee_table has columns EMPTNO, ENAME and DEPTREF. EMPNO is of type "number" and assume that EMPNO has only one row of value "15". ENAME is of type "string" and assume that ENAME has only one row of value "John". DEPTREF is of type "REF" and assume

that DEPTREF has only one row of value, which is a reference to a row of value corresponding to DEPTNO = 1001 in another table called Department_table. Assume that Department_table is in the schema SCOTT and has columns DEPTNO with value "1001" and DEPTNAME with value "SPORTS".

A SQL query such as "SELECT EMPNO, DEPTREF FROM Employee-table produces a SQL result set that maps to the following XML result set:

```
<ROWSET>
  <ROW>
    <EMPNO>15</EMPNO>
    <DEPTREF>0344855FF4ABBCC3333</DEPTREF>
  </ROW>
</ROWSET>
```

According to certain embodiments of the invention, the REF value in the DEPTREF column is converted to XML form for the XML result set by converting the object relational REF value into a binary value that is encoded in hexadecimal ("HEX"). Thus, in the above XML result set "0344855FF4ABBCC3333" is the encoded HEX.

According to certain other embodiments of the invention, the REF value in the DEPTREF column is converted to XML form by converting the REF value into a Database Uniform Resource Indicator ("DBURI") reference. DBURI references are described in detail in U.S. patent application serial number ________________, Attorney Docket No. 50277-1564, entitled "METHOD AND APPARATUS FOR XML VISUALIZATION OF A RELATIONAL DATABASE AND UNIVERSAL RESOURCE IDENTIFIERS TO DATABASE DATA AND METADATA," naming as inventors Muralidhar Krishnaprasad, Viswanathan Krishnamurthy, Ravi Murthy.

Using the same example above, the DBURI reference may appear as, SCOTT/DEPARTMENT_TABLE/ROW[DEPTNO= "1001"], where Deptno is the primary key information. Thus, the XML result set may appear as:

```
<ROWSET>
 <ROW>
   <EMPNO>15</EMPNO>
<DEPTREF>/SCOTT/DEPARTMENT_TABLE/ROW[DEPTNO="1001"]</DEPTREF>
 </ROW>
</ROWSET>
```

According to yet other embodiments of the invention, an object identifier that uniquely identifies the object stored in the REF column can be used in creating the DBURI reference. Using the same example above, the DBURI reference may appear as, SCOTT/DEPARTMENT_TABLE/ROW[SYS_NC_OID$="0asfgd23gfm3423n"]

Thus, the XML result set may appear as:

```
<ROWSET>
 <ROW>
  <EMPNO>15</EMPNO>
<DEPTREF>/Scott/Department_Table/Row[SYS_NC_OID$="0asfgd23gfm3423n"]</DEPTREF
>
 </ROW>
</ROWSET>
```

According to an embodiment of the invention, LOB values may be converted to DBUri references as well. For example, if there is a column in the department table called DEPT_PHOTO, which is a BLOB column, then a DBUri reference can be used to reference the BLOB data instead of in-lining the BLOB data in the XML document.

The XML result may appear as follows for the DEPT_PHOTO column:

```
<ROWSET>
  <ROW>
    <DEPTNO>1001</DEPTNO>
<DEPT_PHOTO>/SCOTT/DEPARTMENT_TABLE/ROW[DEPTNO=1001]/
    DEPT_PHOTO/text()</DEPT_PHOTO>
  </ROW>
</ROWSET>
```

The invention is not limited to generating DBUri references to LOBs and REFs. A DBUri reference can be generated to reference any piece of database data that need not be inlined in the visualized document. For example, nested tables, LONGs and other datatypes may be converted to DBUri references instead of inlining the whole data inside the visualized document. The DBUri reference can be generated by using the primary key information or ROWID of the row in the predicates of the Xpath expression.

MAPPING OF RELATIONAL DATABASE SCHEMAS

Relational database schemas are mapped to XML form by mapping the relational database schemas to their corresponding XML-Schemas. For example, referring to FIG.

3A, in order to produce the XML result set of a SQL query to select EMPNO and ENAME at the single row 302, an XML-Schema is generated. The XML-Schema defines the structure of the XML result set, which is in the form of an XML document. Thus, the XML-Schema that corresponds to the XML result set of a SQL query to select EMPNO and ENAME at the single row 302 is as follows, according to certain embodiments of the invention:

```
        <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="ROWSET">
    <xsd:complexType>
        <xsd:element name="ROW" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
              <xsd:element name="EMPNO" type="xsd:integer" minOccurs="0"/>
              <xsd:element name="ENAME" type="xsd:string" nullable="true"
minOccurs="0"/>
          </xsd:complexType>
        </xsd:element>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

As can be seen from the above example, in addition to defining the structure of the XML result set (XML document), XML-Schemas also define the type of the data and constraints, if any, on the data. For example, the above sample XML-Schema indicates, among other things: 1) the URL, i.e., http://www.w3.org/2000/10/XMLSchema, for the namespace that defines the standards that apply to all XML-Schemas, 2) element ROWSET is of type "complex" and contains ROW elements, 3) the ROW elements are of type "complex" and can occur any number of times in the XML document, 3) the element EMPNO, which is embedded in the ROW element, is of type "integer", and 4) the element ENAME, which is embedded in the ROW element, is of type "string". The nullable attribute indicates whether the element value can be NULL or not. "MinOccurs" indicates the minimum number of occurrences of this element in the resultant document.

Alternatively, XML-Schemas can be "inlined" in its corresponding XML result set (XML document). For example, referring to FIG. 3A, the XML result set of a SQL query to select EMPNO and ENAME at the single row 302, can have inlined in the XML result set its corresponding XML-Schema as follows, according to certain embodiments of the invention:

```
<?xml version = '1.0'?>
<DOCUMENT xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
    <xsd:element name="ROWSET">
      <xsd:complexType>
          <xsd:element name="ROW" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:element name="EMPNO" type="xsd:integer" minOccurs="0"/>
                <xsd:element name="ENAME" type="xsd:string" nullable="true"
minOccurs="0"/>
            </xsd:complexType>
          </xsd:element>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <ROWSET xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="#/DOCUMENT/xsd:schema[not(@targetNamespace
)]"
            <ROW num="1">
      <EMPNO>2100</EMPNO>
      <ENAME>JOHN</ENAME>
    </ROW>
  </ROWSET>
</DOCUMENT>
```

XML NAMESPACES

Object relational data type definitions that are associated with a given relational database schema can be bound to a corresponding XML namespace. An XML namespace is a collection of names that are used in XML documents as element types and attribute names. An XML namespace can be defined by a URL. For example, as explained herein, assume that the relational database has a user-defined type "Address_t", which is a complex object type that comprises 3 scalar attributes, namely CITY, STATE and ZIP. For the purpose of explanation, assume that the complex object type, "Address_t", is defined in relational database schema SCOTT. Thus, "Address_t" can be bound to the XML namespace that is defined by the URL, http://ora.com/SCOTT, for example.

An XML-Schema can contain object relational types from different XML namespaces. To illustrate, assume a relational database schema FOO and a relational database schema PEE is created. Further assume that the relational database schema FOO contains the user-defined type called AddressType, and the relational database schema PEE contains the user-defined type called NameType. The AddressType comprises 3

scalar attributes, namely CITY, STATE and ZIP. The NameType comprises 2 scalar attributes, namely FIRSTNAME and LASTNAME. Assume that an object relational table, EMP2, is created. EMP2 has 2 columns, namely ENAME and EADDR. ENAME contains the name of employee, "James Bond" and is of type NameType. EADDR contains the address of James Bond and is of type AddressType.

The following SQL statements illustrate the creation of database schema FOO, database schema PEE, AddressType, NameType and EMP2 in the relational database:

```
        connect sys/sys as sysdba
grant resource,dba to foo identified by ball;
grant resource,dba to pee identified by wee;
grant execute any type to foo;
grant execute any type to pee;
connect foo/ball
create type AddressType as object (street varchar2(30), city varchar2(30), zip
number(5,0));
/
connect pee/wee
create type NameType as object (FirstName varchar2(30), LastName varchar2(30));
/
create table Emp2
(
 ename  NameType,
 eaddr  foo.AddressType
);
```

The syntax used in the preceding statements is merely illustrative. The actual syntax of SQL statements may vary from implementation to implementation. The present invention is not limited to any particular syntax.

In table EMP2, the column ENAME uses the NameType that is defined for relational database schema PEE while the column EADDR uses the Address Type that is defined for relational database schema FOO.

Thus, the XML-Schema that corresponds to the XML result set of a SQL query such as SELECT * FROM EMP2 will contain separate URLs that define the XML namespace for FOO and for PEE. The XML namespace for FOO will contain the definition for AddressType. The XML namespace for PEE will contain the definition for NameType.

The XML-Schemas that correspond to the XML result set of a SQL query such as SELECT * FROM EMP2 are as follows, according to certain embodiments of the invention:

```
<xsd:schema xmlns:FOO="http://ns.oracle.com/xdb/FOO"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns:PEE="http://ns.oracle.com/xdb/PEE">
  <xsd:element name="ROWSET">
    <xsd:complexType>
      <xsd:element name="ROW" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:element name="ENAME" type="PEE:NAMETYPE" nullable="true"
minOccurs="0"/>
          <xsd:element name="EADDR" type="FOO:ADDRESSTYPE"
nullable="true" minOccurs="0"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<schema targetNamespace="http://ns.oracle.com/xdb/FOO"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:FOO="http://ns.oracle.com/xdb/FOO">
  <complexType name="ADDRESSTYPE">
    <element name="STREET" type="string" nullable="true" minOccurs="0"/>
    <element name="CITY" type="string" nullable="true" minOccurs="0"/>
    <element name="ZIP" type="integer" nullable="true" minOccurs="0"/>
  </complexType>
</schema>
<schema targetNamespace="http://ns.oracle.com/xdb/PEE"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:PEE="http://ns.oracle.com/xdb/PEE">
  <complexType name="NAMETYPE">
    <element name="FIRSTNAME" type="string" nullable="true" minOccurs="0"/>
    <element name="LASTNAME" type="string" nullable="true" minOccurs="0"/>
\ </complexType>
</schema>
```

Alternatively, XML-Schemas can be “inlined” in its corresponding XML result set that contains the target data, namely, the name and address of the employee in EMP2. According to certain embodiments of the invention, the XML result set shows its corresponding XML-schemas “inlined”:

```
<?xml version = '1.0'?>
<DOCUMENT xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
```

```
  <schema targetNamespace="http://ns.oracle.com/xdb/PEE"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:PEE="http://ns.oracle.com/xdb/PEE">
    <complexType name="NAMETYPE">
        <element name="FIRSTNAME" type="string" nullable="true" minOccurs="0"/>
        <element name="LASTNAME" type="string" nullable="true" minOccurs="0"/>
    </complexType>
  </schema>
  <schema targetNamespace="http://ns.oracle.com/xdb/FOO"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:FOO="http://ns.oracle.com/xdb/FOO">
    <complexType name="ADDRESSTYPE">
        <element name="STREET" type="string" nullable="true" minOccurs="0"/>
        <element name="CITY" type="string" nullable="true" minOccurs="0"/>
        <element name="ZIP" type="integer" nullable="true" minOccurs="0"/>
    </complexType>
  </schema>
  <xsd:schema xmlns:FOO="http://ns.oracle.com/xdb/FOO"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns:PEE="http://ns.oracle.com/xdb/PEE">
    <xsd:import targetNamespace="http://ns.oracle.com/xdb/PEE"
schemaLocation="#/DOCUMENT/xsd:schema[@targetNamespace='http://ns.oracle.com/
xdb/PEE']"/>
    <xsd:import targetNamespace="http://ns.oracle.com/xdb/FOO"
schemaLocation="#/DOCUMENT/xsd:schema[@targetNamespace='http://ns.oracle.com/
xdb/FOO']"/>
    <xsd:element name="ROWSET">
      <xsd:complexType>
          <xsd:element name="ROW" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:element name="ENAME" type="PEE:NAMETYPE" nullable="true"
minOccurs="0"/>
                <xsd:element name="EADDR" type="FOO:ADDRESSTYPE"
nullable="true" minOccurs="0"/>
            </xsd:complexType>
          </xsd:element>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <ROWSET xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="#/DOCUMENT/xsd:schema[not(@targetNamespace
)]">
    <ROW num="1">
      <ENAME>
        <FIRSTNAME>James</FIRSTNAME>
        <LASTNAME>Bond</LASTNAME>
      </ENAME>
      <EADDR>
        <STREET>00 secret ave</STREET>
        <CITY>London</CITY>
```

```
    <ZIP>39281</ZIP>
   </EADDR>
  </ROW>
 </ROWSET>
</DOCUMENT>
```

UNIFORM HANDLING OF XML DATA AND SQL DATA

Typically, in a relational database, there are pre-defined relational data types. Examples of typical pre-defined relational datatypes are number, date, string, etc. Object-relational databases may also contain user-defined object types. However, in order to provide uniform handling of XML data and SQL data, a datatype called XMLType is natively defined in the relational database system. The XMLType datatype can be used for storing any type of XML data, be it structured XML data or unstructured XML data.

FIG. 4 is a block diagram that illustrates the XMLType storage architecture. Block 402 is XML data that is to be stored using the XMLType shown at block 406. The XMLType encapsulates the underlying storage mechanisms for XML data. Some of the storage mechanisms are shown in the storage layer 414. In storage layer 414 is shown a LOB storage mechanism 408, an object relational storage mechanism 410, and an "other" storage mechanism 412. "Other" storage mechanism 412 can be any appropriate storage mechanism for XML data. Thus, the storage mechanisms in storage layer 414 do not comprise the exhaustive set of storage mechanisms for XML data. Because the XMLType encapsulates the underlying storage mechanisms, a user of XML data only sees the XMLType while the actual details of the underlying storage mechanism of the XML data are hidden from the user. Typically, an administrator of the relational database system chooses the underlying storage mechanism for the XML data that is to be stored. The administrator chooses the type of underlying storage mechanism based on performance considerations of the storage mechanism.

To illustrate storage, some of the fields of a given XML document may contain structured data. Structured data is data that is capable of being mapped to relational columns in the relational database. In another example, assume that the XML document contains the textual content of an entire book. Rather than exploding such an XML document by mapping every element or field of the XML document to a relational column, only fields that contain data that are likely to be queried by a user are mapped to pre-defined relational types, such as a string, number, etc, while the rest of the data may be lumped together and mapped to one column for Character Large Objects (CLOBs). A

user may create his or her own template to specify what fields are to be mapped to relational columns and what fields are to be mapped to CLOBs.

FIG. 5 is a block diagram that illustrates the structure of an XMLType column. Column 504 is an XMLType column that contains an XML document that comprises structured data. The fields of the structured data are mapped into hidden columns 506. For example, the PONO element of the XML document is mapped to a NUMBER column, and PNAME is mapped to a string column. In FIG. 5, the crosshatched area indicates that the columns are hidden from the user's view. The user only sees a single XMLType column.

UNIFORM QUERY INTERFACE

According to certain embodiments of the invention, a uniform query interface is provided in order to define a core set of operations on XMLType data that is stored in the relational database. Such operations are independent of the underlying storage format. According to certain embodiments of the invention, the operations on the XMLType data is functionally abstracted as follows:

1) Extract fragments of a given XML document

2) Test for existence of certain structures in the XML document

3) Extract certain data values in the XML document

4) Transforming a given XML document. The above list of operations is not an exhaustive list of operations for XMLType data. To illustrate some of the operations, assume that an XML document called "X" contains purchase order data. The purchase order data includes a purchase order number "PONO" with value "21", purchase order name "PNAME" with value "JOHN" and a collection of lineitems and appears as follows:

```
<PO>
  <PONO> 21</PONO>
  <PNAME>JOHN</PNAME>
  <LINEITEM>
    <LINEITEMNO> 100    </LINEITEMNO>
   ......
      </LINEITEM>
  <! Other lineitems >
</PO>
```

Thus, according to certain embodiments of the invention, an example of the operation for extracting a fragment of XML document "X" is as follows:

EXTRACT (X, 'PO/LINEITEM')

The above operation extracts a fragment of the document X, wherein the fragment is a sub tree comprising of all the branches under LINEITEM.

An example of the operation for extracting certain data values in the XML document "X" is as follows:

EXTRACTVALUE (X, 'PO/PNAME')

The above operation extracts the scalar value in PNAME, namely, "JOHN".

An example of the operation for testing for the existence of certain elements in the XML document "X" is as follows:

EXISTSNODE (X, 'PO/[PONO=21]')

The above operation tests whether the XML document "X" has an element called PO, which in turn has a child called PONO whose value is 21.

An example of the operation for transforming the XML document using a XSL stylesheet is as follows:

TRANSFORM (X, '<xsl> ......stylesheet ..... </xsl>')

The above operations are completely agnostic as to the underlying storage format of the XML document.

QUERY REWRITES

According to certain embodiments of the invention, a mechanism is provided for rewriting user queries into a form that leverages the data access and data manipulation capabilities of the underlying storage mechanism in the relational database.

The fields of structured data of a given XML document may be mapped into separate relational columns if the data in the fields is likely to be frequently queried by a user. For example, assume that one of the fields of an XML document contains an employee name, and that it is anticipated that users will frequently query the XML documents based on the employee name value. Under these conditions, employee names may be stored in a relational column called ENAME that is separate from the column that stores the XML document itself. If a XML user submits a query to access XML documents based on a particular employee's name, then the XML user's query is automatically rewritten to access only the ENAME column.

In contrast, if no query rewrite mechanism is provided and employee names are not stored in a separate column, then, when the XML user submits a query to access a XML documents based on a particular employee's name, a Document Object Model

(DOM) is created for each XML document by parsing the XML document. Then the search is performed on the DOM for the employee's name by applying an appropriate XPATH expression. Creating a DOM and then performing a search on the DOM is clearly less efficient.

In another example, the existing indexing capabilities of the relational database are used to satisfy the XML user's query. If data is expected to be frequently queried, the data can be stored in a separate relational column, and a B-tree index may be built on that column. Then, if an XML user query is submitted to select, for example, a row where PONO=21, then the B-tree index on the PONO column can be used to identify the rows that contain XML documents that have the value 21 in the PONO column. Similarly, if an XML document is stored as a LOB, then a text index may be used to optimize a search on the LOB column.

A mechanism is provided in the database to generate a database query, for example a SQL query, based on the user's XML query and the mapping information associated with the storage of the XML document in the relational database.

Referring to FIG. 5, assume that the table of FIG. 5 is called PO-TABLE and that a B-tree index is generated on the relational column PONO. Assume that an XML user submits the following query:

```
SELECT * From PO-TABLE
Where EXISTNODE (PO-XML, '/PO[PONO=21]')
```

According to certain embodiments of the invention, the above XML user's query is converted to the following:

```
SELECT * From PO-TABLE
Where PO-XML .PONO=21
```

Thus, an index search can be performed for the predicate of PONO.

HARDWARE

FIG. 6 is a block diagram that illustrates a computer system 600 upon which an embodiment of the invention may be implemented. Computer system 600 includes a bus 602 or other communication mechanism for communicating information, and a processor

604 coupled with bus 602 for processing information. Computer system 600 also includes a main memory 606, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 602 for storing information and instructions to be executed by processor 604. Main memory 606 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 604. Computer system 600 further includes a read only memory (ROM) 608 or other static storage device coupled to bus 602 for storing static information and instructions for processor 604. A storage device 610, such as a magnetic disk or optical disk, is provided and coupled to bus 602 for storing information and instructions.

Computer system 600 may be coupled via bus 602 to a display 612, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 614, including alphanumeric and other keys, is coupled to bus 602 for communicating information and command selections to processor 604. Another type of user input device is cursor control 616, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 604 and for controlling cursor movement on display 612. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

The invention is related to the use of computer system 600 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are implemented by computer system 600 in response to processor 604 executing one or more sequences of one or more instructions contained in main memory 606. Such instructions may be read into main memory 606 from another computer-readable medium, such as storage device 610. Execution of the sequences of instructions contained in main memory 606 causes processor 604 to perform the process steps described herein. One or more processors in a multi-processing arrangement may also be employed to execute the sequences of instructions contained in main memory 606. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 604 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks,

such as storage device 610. Volatile media includes dynamic memory, such as main memory 606. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 602. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 604 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 600 can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector coupled to bus 602 can receive the data carried in the infrared signal and place the data on bus 602. Bus 602 carries the data to main memory 606, from which processor 604 retrieves and executes the instructions. The instructions received by main memory 606 may optionally be stored on storage device 610 either before or after execution by processor 604.

Computer system 600 also includes a communication interface 618 coupled to bus 602. Communication interface 618 provides a two-way data communication coupling to a network link 620 that is connected to a local network 622. For example, communication interface 618 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 618 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 618 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 620 typically provides data communication through one or more networks to other data devices. For example, network link 620 may provide a connection

through local network 622 to a host computer 624 or to data equipment operated by an Internet Service Provider (ISP) 626. ISP 626 in turn provides data communication services through the worldwide packet data communication network now commonly referred to as the "Internet" 628. Local network 622 and Internet 628 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 620 and through communication interface 618, which carry the digital data to and from computer system 600, are exemplary forms of carrier waves transporting the information.

Computer system 600 can send messages and receive data, including program code, through the network(s), network link 620 and communication interface 618. In the Internet example, a server 630 might transmit a requested code for an application program through Internet 628, ISP 626, local network 622 and communication interface 618. In accordance with the invention, one such downloaded application implements the techniques described herein.

The received code may be executed by processor 604 as it is received, and/or stored in storage device 610, or other non-volatile storage for later execution. In this manner, computer system 600 may obtain application code in the form of a carrier wave.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A method for accessing data within a relational database, the method comprising the steps of:

receiving, at a database server, a request that includes information in XML syntax, wherein the information specifies target data that resides in one or more portions of the relational database;

determining, based on the information, the one or more portions of the relational database that correspond to the target data; and

retrieving the target data from the one or more portions of the relational database.

2. The method of Claim 1, further comprising the step of, prior to receiving the request, providing an XML visualization of portions of the relational database based on access privileges of a user who is issuing the request.

3. The method of Claim 1, further comprising providing an XML visualization of portions of the relational database in response to the request.

4. The method of Claim 1, further comprising performing operations on the target data after retrieving the target data and wherein the operations on the target data are performed by issuing an Xpath notation to traverse an XML visualization of the one or more portions of the relational database.

5. The method of Claim 4, wherein the operations on the target data comprise updating, deleting, reading, and inserting.

6. The method of Claim 2, wherein providing the XML visualization comprises dynamically generating one or more XML schema based on the access privileges that the user has at a time of generating the one or more XML schema.

7. The method of Claim 1, wherein the request is an Xpath query.

8. The method of Claim 7, wherein the Xpath query is mapped into a SQL query.

9. The method of Claim 7, wherein a translator of the Xpath query directly generates execute-time structures without being mapped into an SQL query.

10. The method of Claim 1, further comprising the step of creating a reference to the one or more objects of the relational database by using uniform resource identifier reference within the relational database.

11. A method for accessing data in a relational database, the method comprising the steps of:

establishing a mapping between Uniform Resource Locators (URLs) and data items within the relational database based on where said data items reside within a schema of the relational database; and

locating the data items within the relational database based on the Uniform Resource Locators and the mapping.

12. The method of Claim 11, wherein the step of locating includes:

a database server receiving a URL that is mapped to a particular data item; and

the database server determining where the data item resides in the schema based on the URL and the mapping.

13. The method of Claim 11, wherein the step of locating includes:

an entity that resides outside a database server receiving a URL that is mapped to a particular data item;

the entity determining where the data item resides in the schema based on the URL and the mapping; and

the entity sending a message to the database server to cause the database server to access the data item, wherein the message includes information about where said data item resides in said schema.

14. The method of Claim 11, further comprising the step of storing the URLs in a column of a table that stores the data items, wherein the URL that maps to a particular data item is stored in said column within a row associated with that particular data item.

15. The method of Claim 14, wherein one or more of the URLs include information indicating a data type of the data item associated with the URLs.

16. The method of Claim 14, wherein:

one or more of the URLs stored in the column do not have portions that indicate certain information about where the corresponding data items reside;

the method further includes a database server adding the portions to said one or more URLs prior to providing said one or more URLs to any entity outside said database server; and

the database server determines data type information of data items that are associated with the one or more URLs by accessing auxiliary information that is stored with the corresponding URL.

17. The method of Claim 11, wherein:

the step of locating is performed by a database server; and

the method further includes the step of the database server generating output in a form XML documents that include said data items.

18. A method for accessing a data item stored in a relational database, the method comprising the steps of:

generating, within a database server that manages the relational database, a Uniform Resource Locator (URL) for the data item based on where the data item resides within the relational database;

providing the URL to an entity that resides outside the database server;

receiving, at the database server, the URL; and

in response to receiving the URL, resolving the URL within the database server to locate the data item.

19. The method of Claim 18, wherein the step of generating the URL includes adding to the URL data that indicates a data type associated with the data item.

20. A computer-readable medium carrying instructions for accessing data within a relational database, the computer-readable medium comprising instructions for performing the steps of:

receiving, at a database server, a request that includes information in XML syntax, wherein said information specifies target data that resides in one or more portions of the relational database;

determining, based on the information, the one or more portions of the relational database that correspond to the target data; and

retrieving the target data from the one or more portions of the relational database.

21. The computer-readable medium of Claim 20, further comprising instructions for performing the step of, prior to receiving the request, providing an XML visualization of portions of the relational database based on access privileges of a user who is issuing the request.

22. The computer-readable medium of Claim 20, further comprising instructions for providing an XML visualization of portions of the relational database in response to the request.

23. The computer-readable medium of Claim 20, further comprising instructions for performing operations on the target data after retrieving the target data and wherein the operations on the target data are performed by issuing an Xpath notation to traverse an XML visualization of the one or more portions of the relational database.

24. The computer-readable medium of Claim 23, wherein the operations on the target data comprise updating, deleting, reading, and inserting.

25. The computer-readable medium of Claim 21, wherein providing the XML visualization comprises dynamically generating one or more XML schema based on the access privileges that the user has at a time of generating the one or more XML schema.

26. The computer-readable medium of Claim 20, wherein the request is an Xpath query.

27. The computer-readable medium of Claim 26, wherein the Xpath query is mapped into a SQL query.

28. The computer-readable medium of Claim 26, wherein a translator of the Xpath query directly generates execute-time structures without being mapped into an SQL query.

29. The computer-readable medium of Claim 20, further comprising instructions for performing the step of creating a reference to the one or more objects of the relational database by using uniform resource identifier reference within the relational database.

30. A computer-readable medium carrying instructions for accessing data in a relational database, the computer-readable medium comprising instructions for performing the steps of:

establishing a mapping between Uniform Resource Locators (URLs) and data items within the relational database based on where said data items reside within a schema of the relational database; and

locating the data items within the relational database based on the Uniform Resource Locators and the mapping.

31. The computer-readable medium of Claim 30, wherein the step of locating includes:

a database server receiving a URL that is mapped to a particular data item; and

the database server determining where the data item resides in the schema based on the URL and the mapping.

32. The computer-readable medium of Claim 30, wherein the step of locating includes:

an entity that resides outside a database server receiving a URL that is mapped to a particular data item;

the entity determining where the data item resides in the schema based on the URL and the mapping; and

the entity sending a message to the database server to cause the database server to access the data item, wherein the message includes information about where said data item resides in said schema.

33. The computer-readable medium of Claim 30, further comprising instructions for performing the step of storing the URLs in a column of a table that stores the data items, wherein the URL that maps to a particular data item is stored in said column within a row associated with that particular data item.

34. The computer-readable medium of Claim 33, wherein one or more of the URLs include information indicating a data type of the data item associated with the URLs.

35. The computer-readable medium of Claim 33, wherein:

one or more of the URLs stored in the column do not have portions that indicate certain information about where the corresponding data items reside;

the method further includes a database server adding the portions to said one or more URLs prior to providing said one or more URLs to any entity outside said database server; and

the database server determines data type information of data items that are associated with the one or more URLs by accessing auxiliary information that is stored with the corresponding URL.

36. The computer-readable medium of Claim 30, wherein:

the step of locating is performed by a database server; and

the computer-readable medium further includes instructions for performing the step of the database server generating output in a form XML documents that include said data items.

37. A computer-readable medium carrying instructions for accessing a data item stored in a relational database, the computer-readable medium comprising instructions for performing the steps of:

generating, within a database server that manages the relational database, a Uniform Resource Locator (URL) for the data item based on where the data item resides within the relational database;

providing the URL to an entity that resides outside the database server;

receiving, at the database server, the URL; and

in response to receiving the URL, resolving the URL within the database server to locate the data item.

38. The computer-readable medium of Claim 37, wherein the step of generating the URL includes adding to the URL data that indicates a data type associated with the data item.

39. A method of providing data, the method comprising the steps of:

receiving, at a database server, a relational database query to retrieve data that resides in a relational database; and

the database server responding to the relational database query by providing a result of the relational database query from the relational database in a form of one or more XML documents.

40. The method of Claim 39, wherein the relational database query specifies which values that answer the relational database query are to be contained in the one or more XML documents, and which other values that answer the relational database query are to be referenced in the one or more XML documents by reference links.

41. The method of Claim 39, wherein the step of providing a result to the relational database query in a form of one or more XML documents comprises providing an XML document in which some values that are selected by the relational database query are contained in the XML document and other values that are selected by the relational database query are referenced in the XML document.

42. The method of Claim 39 further comprising the step of, during a processing of the relational database query, generating one or more XML-schema that describe features of the one or more XML documents, wherein the features comprise structure of the one or more XML documents.

43. The method of Claim 42 further comprising storing at least one of the one or more XML-schema within at least one of the one or more XML documents.

44. The method of Claim 39 further comprising the step of, during a processing of the relational database query, generating one or more XML -schema that describe features of the one or more XML documents, wherein the features comprise constraints of the one or more XML documents.

45. The method of Claim 42, wherein the one or more XML-schema corresponds to metadata that is associated with the data that is retrieved in response to the relational database query.

46. The method of Claim 42, wherein the one or more XML-schema comprises relational database object types from one or more XML namespaces, wherein one or more relational database schemas are mapped to each of the one or more XML namespaces.

47. The method of Claim 39, further comprising generating the one or more XML documents, and wherein the step of generating the one or more XML documents comprises using a set of rules that is stored in the relational database to determine how to structure data contained in the one or more XML documents.

48. The method of Claim 39, wherein the relational database query selects a plurality of rows in the relational database and wherein the one or more XML documents includes an XML document that aggregates data from the plurality of rows.

49. The method of Claim 47, wherein the set of rules comprises rules for mapping a relational database object to a corresponding Database Uniform Resource Indicator reference.

50. The method of Claim 47, wherein the set of rules comprises rules for mapping a query language reference to a corresponding Database Uniform Resource Indicator reference.

51. The method of Claim 47, wherein the set of rules comprises rules for mapping relational data that is associated with a LOB column to a corresponding Database Uniform Resource Indicator reference.

52. The method of Claim 39, wherein the relational database query is satisfied by a plurality of rows and the one or more XML documents includes a single XML document for said plurality of rows.

53. A computer-readable medium carrying one or more sequences of instructions of providing data, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps:

receiving, at a database server, a relational database query to retrieve data that resides in a relational database; and

the database server responding to the relational database query by providing a result of the relational database query from the relational database in a form of one or more XML documents.

54. The computer-readable medium of Claim 53, wherein the relational database query specifies which values that answer the relational database query are to be contained in the one or more XML documents, and which other values that answer the relational database query are to be referenced in the one or more XML documents by reference links.

55. The computer-readable medium of Claim 53, wherein the step of providing a result to the relational database query in a form of one or more XML documents comprises providing an XML document in which some values that are selected by the relational database query are contained in the XML document and other values that are selected by the relational database query are referenced in the XML document.

56. The computer-readable medium of Claim 53 further comprising the step of, during a processing of the relational database query, generating one or more XML-schema that describe features of the one or more XML documents, wherein the features comprise structure of the one or more XML documents.

57. The computer-readable medium of Claim 56 further comprising storing at least one of the one or more XML-schema within at least one of the one or more XML documents.

58. The computer-readable medium of Claim 53 further comprising the step of, during a processing of the relational database query, generating one or more XML - schema that describe features of the one or more XML documents, wherein the features comprise constraints of the one or more XML documents.

59. The computer-readable medium of Claim 56, wherein the one or more XML-schema corresponds to metadata that is associated with the data that is retrieved in response to the relational database query.

60. The computer-readable medium of Claim 56, wherein the one or more XML-schema comprises relational database object types from one or more XML namespaces, wherein one or more relational database schemas are mapped to each of the one or more XML namespaces.

61. The computer-readable medium of Claim 53, further comprising generating the one or more XML documents, and wherein the step of generating the one or more XML documents comprises using a set of rules that is stored in the relational database to determine how to structure data contained in the one or more XML documents.

62. The computer-readable medium of Claim 53, wherein the relational database query selects a plurality of rows in the relational database and wherein the one or more XML documents includes an XML document that aggregates data from the plurality of rows.

63. The computer-readable medium of Claim 61, wherein the set of rules comprises rules for mapping a relational database object to a corresponding Database Uniform Resource Indicator reference.

64. The computer-readable medium of Claim 61, wherein the set of rules comprises rules for mapping a query language reference to a corresponding Database Uniform Resource Indicator reference.

65. The computer-readable medium of Claim 61, wherein the set of rules comprises rules for mapping relational data that is associated with a LOB column to a corresponding Database Uniform Resource Indicator reference.

66. The computer-readable medium of Claim 53, wherein the relational database query is satisfied by a plurality of rows and the one or more XML documents includes a single XML document for said plurality of rows.

67. A method for managing data in a relational database, the method comprising the steps of:

receiving at a database server an XML document for storage in the relational database;

storing in the relational database mapping information that indicates a mapping of one or more fields in the XML document to corresponding columns in the relational database;

storing data from the XML document in the relational database in locations that are determined based on the mapping information; and

in response to the database server receiving a request for data from the XML document, inspecting the mapping information to determine how to access the data from the XML document.

68. The method of Claim 67 further comprising the steps of:

determining which corresponding columns in the relational database to use to store data from fields of the XML document; and

generating the mapping information of the one or more fields in the XML document to corresponding columns in the relational database.

69. The method of Claim 67, wherein the step of storing data includes the steps of:

storing values for a plurality of fields of the XML document into a first column of a relational table; and

storing values for at least one field of the XML document into a second column of the relational table, wherein the second column is different from said first column.

70. The method of Claim 67, wherein a request for data is an XML query.

71. The method of Claim 70, further comprising the steps of:

generating a database query based on the XML query and the mapping information; and

executing the database query to access data from the XML document.

72. The method of Claim 71, wherein the database query directly accesses the underlying relational columns and uses indexes.

73. The method of Claim 72, wherein the indexes are relational indexes.

74. The method of Claim 72, wherein the indexes are text indexes.

75. The method of Claim 67, wherein a structure of the XML document is defined by a user to contain one or more fields of structured data, and wherein structured data is data that is of a type that corresponds to an existing relational data object type in the relational database.

76. The method of Claim 67, wherein a structure of the XML document is defined by a user to contain one or more fields of unstructured data, and wherein unstructured data is data that is mapped to a LOB data type in the relational database during processing of the XML document for storage in the relational database.

77. The method of Claim 67, wherein the mapping information is defined by a user.

78. The method of Claim 77, wherein the user has an option to map more than one field of the XML document to each column of the relational database for storage.

79. The method of Claim 67, wherein an XMLType datatype is defined by a database server that manages the relational database for storing the XML document.

80. The method of Claim 79, wherein a set of operations are defined for the XMLType datatype by the database server.

81. The method of Claim 80, wherein the set of operations comprises.
extracting fragments of the XML document;
testing for existence of one or more elements in the XML document;
extracting one or more values from the XML document; and
transforming the XML document.

82. The method of Claim 67, wherein:

the mapping information maps a field of said XML document to a plurality of columns of a relational table; and

the step of storing data from the XML document includes parsing said field of said XML document to identify a plurality of values and storing each of the plurality of values in a different one of said plurality of columns of the relational table.

83. The method of Claim 67, wherein:

the request specifies criteria for a field of the XML document;

the method includes determining whether the criteria is satisfied by performing the steps of:

determining, based on said mapping information, that the field is one of a plurality of fields that is mapped to a particular column;

parsing data in said particular column to locate a value of said field for said XML document; and

determining whether said value satisfies the criteria.

84. A computer-readable medium carrying one or more sequences of instructions for managing data within a relational database, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving at a database server an XML document for storage in the relational database;

storing in the relational database mapping information that indicates a mapping of one or more fields in the XML document to corresponding columns in the relational database;

storing data from the XML document in the relational database in locations that are determined based on the mapping information; and

in response to the database server receiving a request for data from the XML document, inspecting the mapping information to determine how to access the data from the XML document.

85. The computer-readable medium of Claim 84 further comprising the steps of:

determining which corresponding columns in the relational database to use to store data from fields of the XML document; and

generating the mapping information of the one or more fields in the XML document to corresponding columns in the relational database.

86. The computer-readable medium of Claim 84, wherein the step of storing data includes the steps of:

storing values for a plurality of fields of the XML document into a first column of a relational table; and

storing values for at least one field of the XML document into a second column of the relational table, wherein the second column is different from said first column.

87. The computer-readable medium of Claim 84, wherein a request for data is an XML query.

88. The computer-readable medium of Claim 87, further comprising the steps of:

generating a database query based on the XML query and the mapping information; and

executing the database query to access data from the XML document.

89. The computer-readable medium of Claim 88, wherein the database query directly accesses the underlying relational columns and uses indexes.

90. The computer-readable medium of Claim 89, wherein the indexes are relational indexes.

91. The computer-readable medium of Claim 89, wherein the indexes are text indexes.

92. The computer-readable medium of Claim 84, wherein a structure of the XML document is defined by a user to contain one or more fields of structured data, and wherein structured data is data that is of a type that corresponds to an existing relational data object type in the relational database.

93. The computer-readable medium of Claim 84, wherein a structure of the XML document is defined by a user to contain one or more fields of unstructured data, and wherein unstructured data is data that is mapped to a LOB data type in the relational database during processing of the XML document for storage in the relational database.

94. The computer-readable medium of Claim 84, wherein the mapping information is defined by a user.

95. The computer-readable medium of Claim 94, wherein the user has an option to map more than one field of the XML document to each column of the relational database for storage.

96. The computer-readable medium of Claim 84, wherein an XMLType datatype is defined by a database server that manages the relational database for storing the XML document.

97. The computer-readable medium of Claim 96, wherein a set of operations are defined for the XMLType datatype by the database server.

98. The computer-readable medium of Claim 97, wherein the set of operations comprises.

extracting fragments of the XML document;

testing for existence of one or more elements in the XML document;

extracting one or more values from the XML document; and

transforming the XML document.

99. The computer-readable medium of Claim 84, wherein:

the mapping information maps a field of said XML document to a plurality of columns of a relational table; and

the step of storing data from the XML document includes parsing said field of said XML document to identify a plurality of values and storing each of the plurality of values in a different one of said plurality of columns of the relational table.

100. The computer-readable medium of Claim 84, wherein:

the request specifies criteria for a field of the XML document;

the method includes determining whether the criteria is satisfied by performing the steps of:

determining, based on said mapping information, that the field is one of a plurality of fields that is mapped to a particular column;

parsing data in said particular column to locate a value of said field for said XML document; and

determining whether said value satisfies the criteria.

SCHEMA 102	OBJECTS 104
SCOTT	EMP
JONES	DEPT
• • •	• • •

FIG. 1A

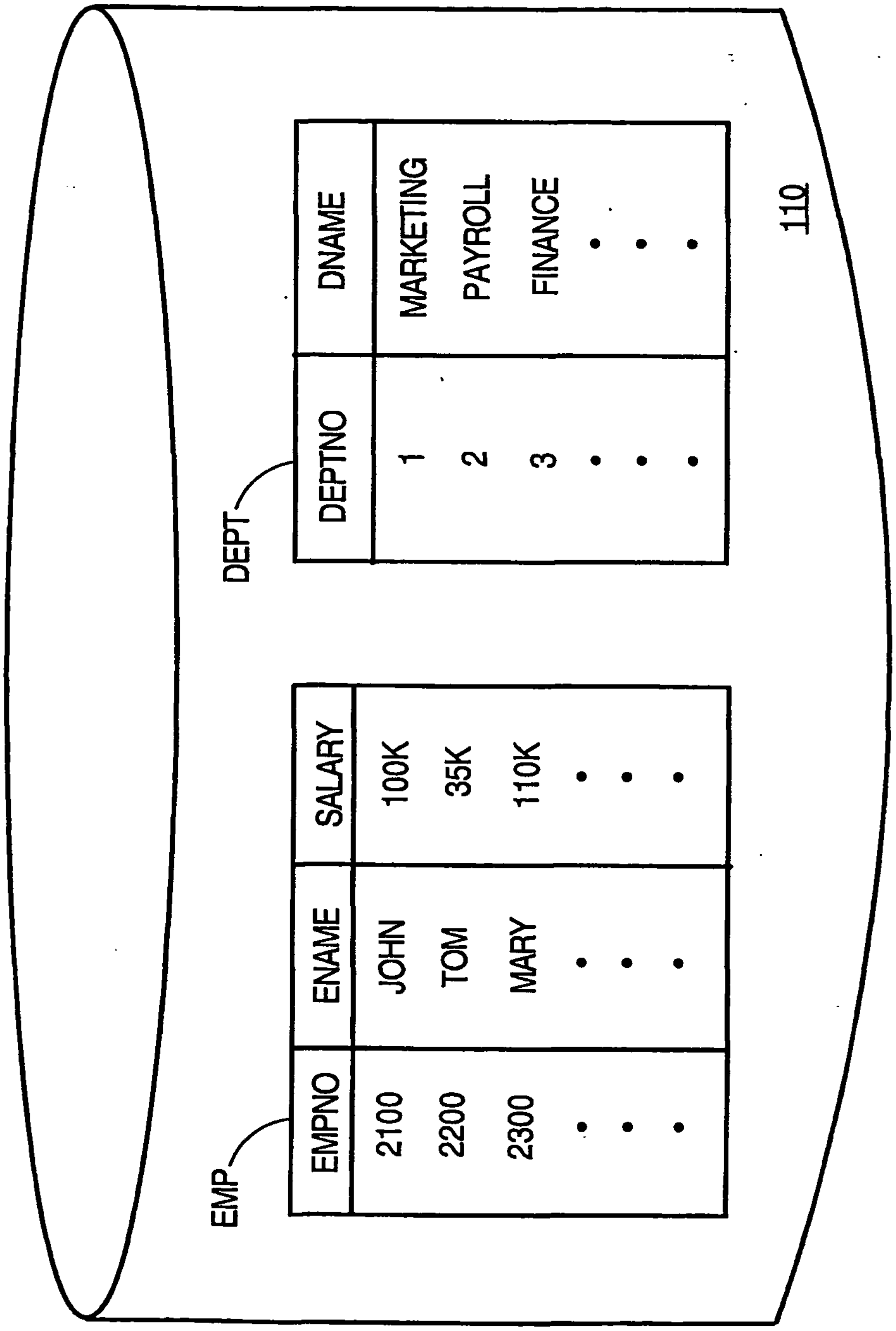

EMP
EMPNO
ENAME
SALARY
2100
JOHN
100K
2200
TOM
35K
2300
MARY
110K
DEPT
DEPTNO
DNAME
1
MARKETING
2
PAYROLL
3
FINANCE
110

FIG. 1B

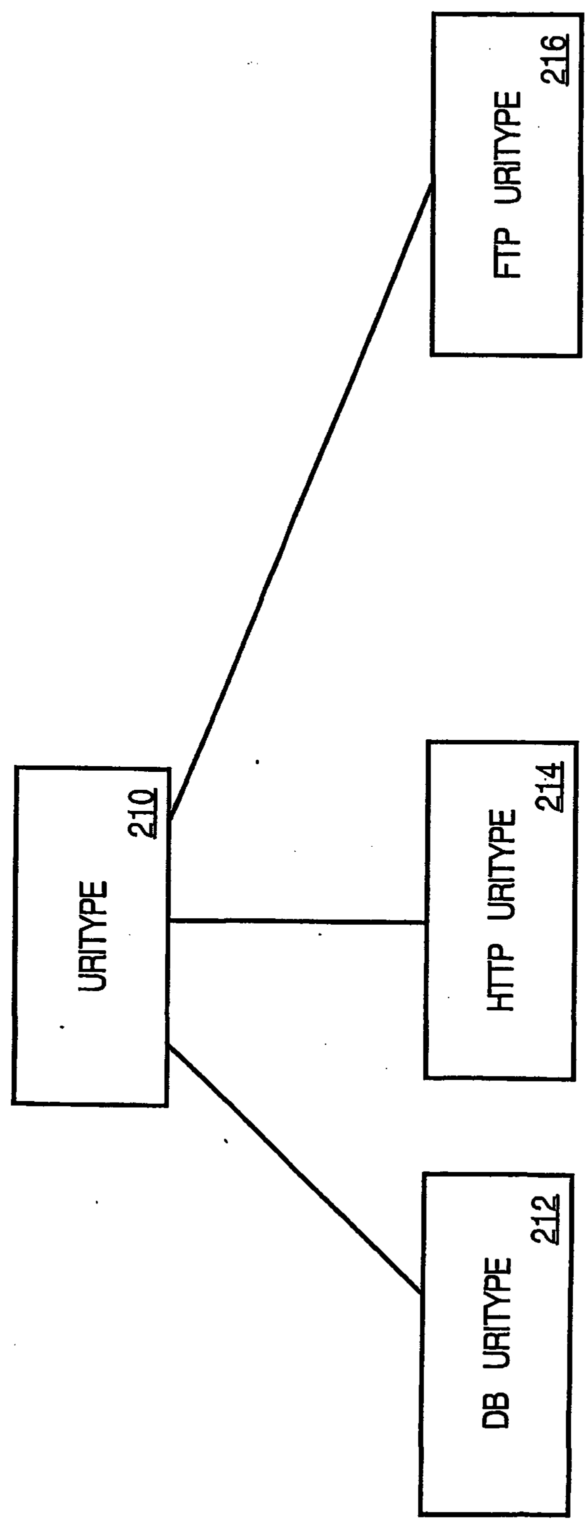


FIG. 2A

PURCHASE ORDER TABLE 200

	PURCHASE ORDER NUMBER 250	PURCHASE ORDER LINK 260
271	1	http: //machineA/serveletA/db1/user1/table1/row[PO NO=1000]/PO
272	2	ftp: //machineB/serveletB/db2/user2/table2/row[PO NO=2000]/PO
273	3	/Tom/Table3/row[PO NO=3000]/PO

FIG. 2B

EMP

	EMPNO	ENAME	SALARY
302	2100	JOHN	55K
304	2200	MARY	65K
	• • •	• • •	• • •

FIG. 3A

EMP

	EMPNO	ENAME
310	2100	JOHN
312	2200	MARY
	⋮	⋮

FIG. 3B

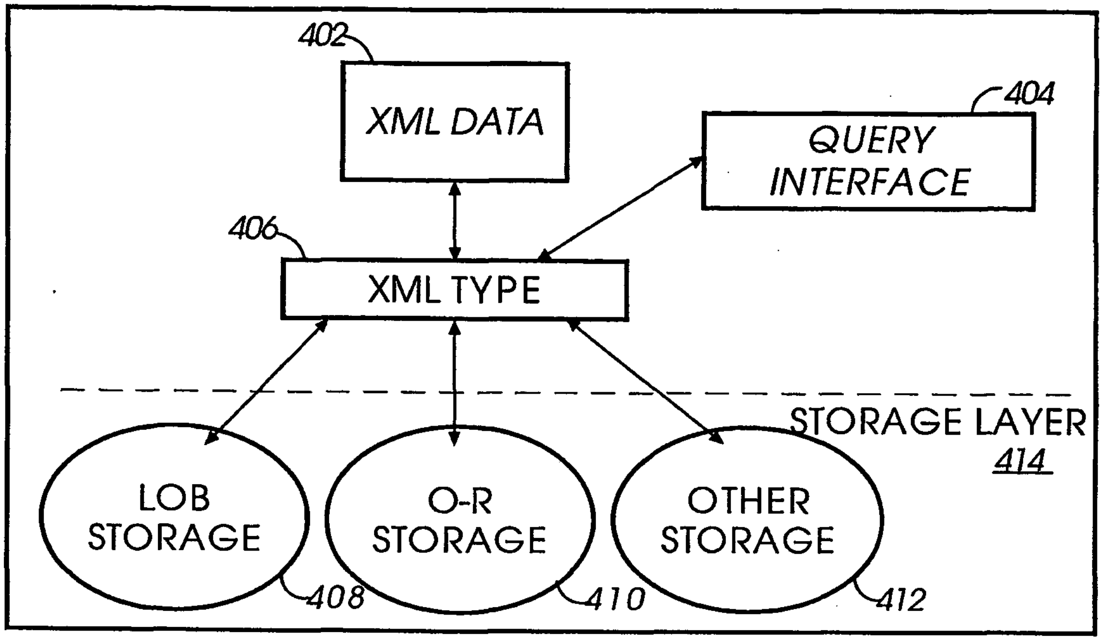

402
XML DATA
404
QUERY INTERFACE
406
XML TYPE
STORAGE LAYER
414
LOB STORAGE
408
O-R STORAGE
410
OTHER STORAGE
412

FIG. 4

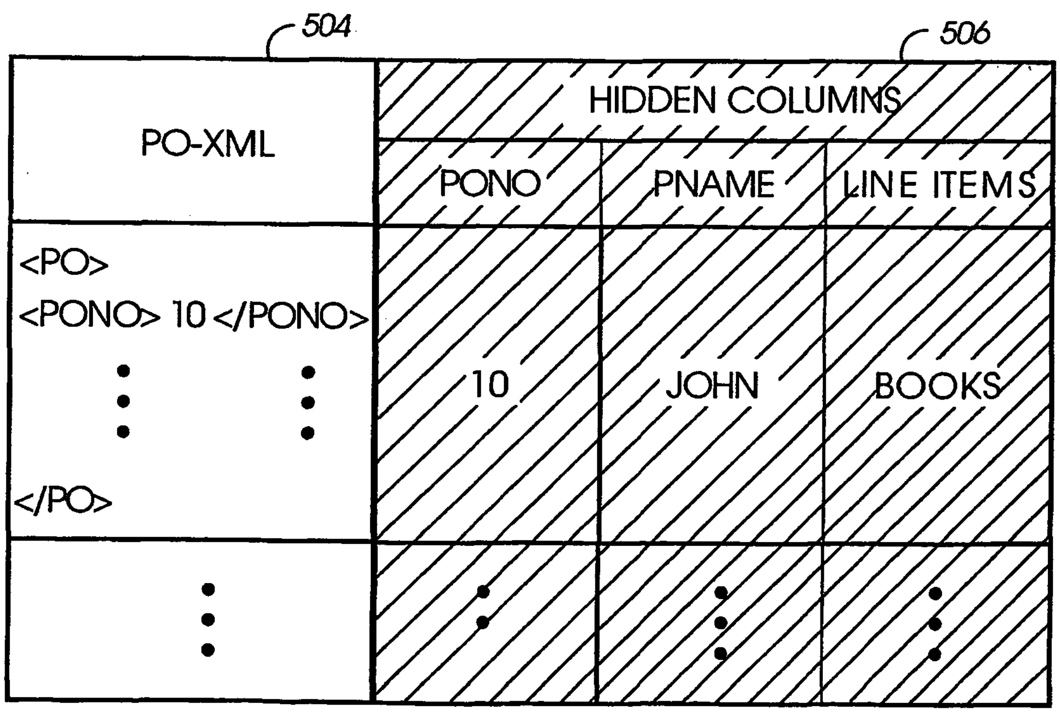

504
506
PO-XML
HIDDEN COLUMNS
PONO
PNAME
LINE ITEMS
<PO>
<PONO> 10 </PONO>
</PO>
10
JOHN
BOOKS

FIG. 5

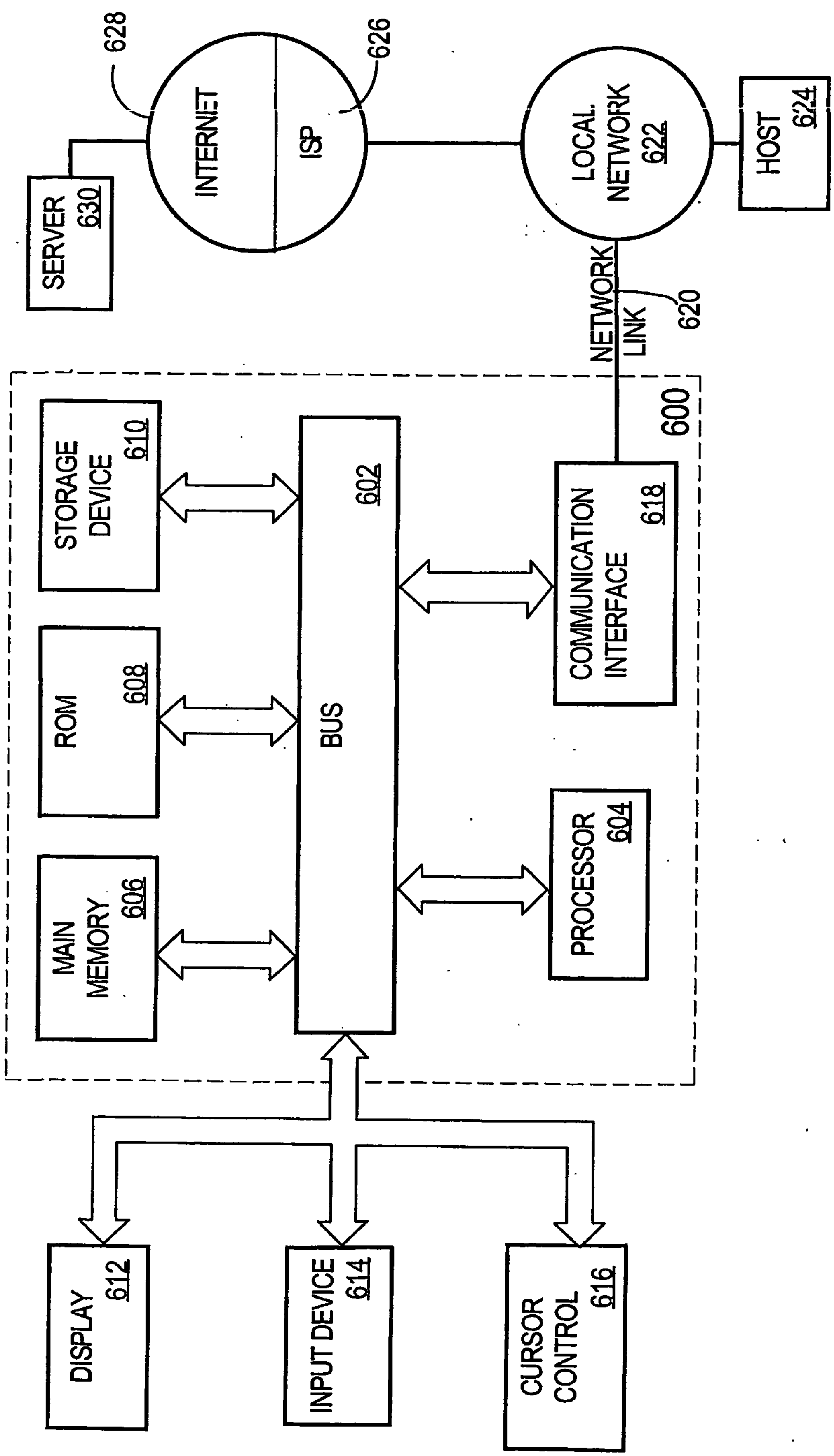

SERVER
630
INTERNET
628
ISP
626
LOCAL NETWORK
622
HOST
624
NETWORK LINK
620
600
STORAGE DEVICE
610
ROM
608
MAIN MEMORY
606
BUS
602
COMMUNICATION INTERFACE
618
PROCESSOR
604
DISPLAY
612
INPUT DEVICE
614
CURSOR CONTROL
616

FIG. 6